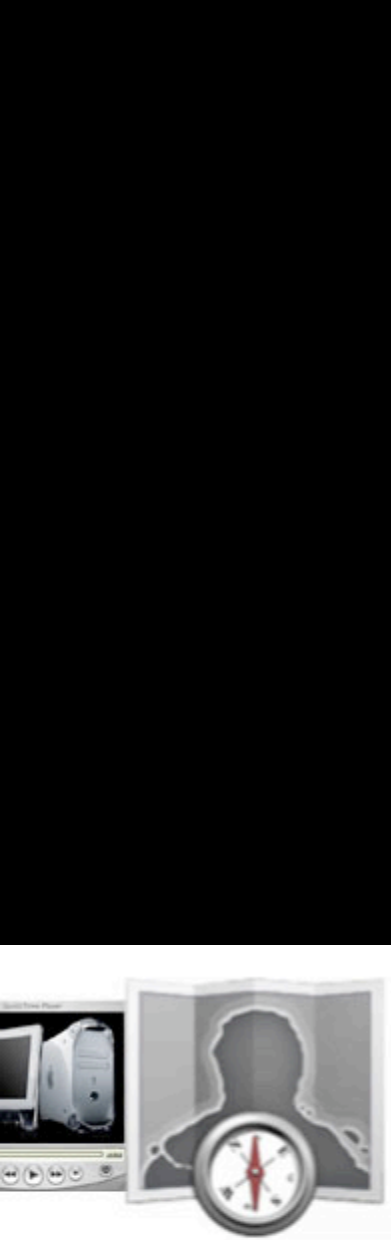


Drawing With Cocoa

Session 305

Drawing With Cocoa

John C. Randolph
Sr. Software Engineer
Apple Worldwide Developer Relations

What We'll Cover

- How to draw using the Cocoa frameworks
- The Cocoa classes involved in 2D rendering
- Manipulating the coordinate space
- Images, paths, and text
- How the Quartz drawing model is reflected in the Cocoa framework

The Quartz 2D Imaging Model

- Familiar from Postscript, PDF, SVG
- Paths are described in terms of instructions for moving a pen in a two-dimensional space
- Paths consist of elements including moves, lines, curves, and character glyphs
- Paths can be stroked with a given line width, filled with a pattern, stroked with a dashed line style, etc.

The Quartz 2D Imaging Model (Cont.)

- Alpha-channel (transparency) support
  - Can use pre-multiplied images
- Rich set of compositing operators
  - 14 different compositing modes
  - Uses both *source* and *destination* alpha
  - Use CompositeLab to try them out

Some Good Advice

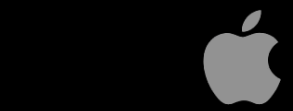
- In Cocoa development, concentrate on the code that is unique to your application
- Do not reinvent the wheel
- Change the environment to make your work easier
- As a friend of mine has said on many occasions . . .

“Use the Kit, Luke!”

- First, see if there is an easy way to do what you want to do
- Many common drawing tasks can be done with high-level AppKit classes
- Paths, colors, text, images, and coordinate transformations all have associated AppKit classes

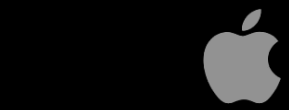
Where Do I Draw?

- Drawing destination
  - Objects that implement **-lockFocus** and **-unlockFocus**
- NSView
  - A “Rectangle of Responsibility” in a window
- NSImage
  - A high-level, resolution-independent abstraction of images



NSView

- Provides a class for drawing, printing and handling events
- An NSView has its own coordinate space
- NSViews are arranged in a hierarchy within a window



What Is NSImage?

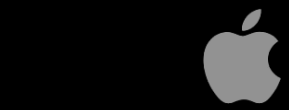
- A high-level abstraction for images
- A container for NSImageRep instances
- Both a *thing* to draw, and a *place* to draw
- Use NSImage to load .tiff, .jpeg, .gif, .pdf, and other image file types
- Use NSImage as a destination for off-screen drawing

Demo

Tinted Image

What I Did Wrong in That Demo

- I was not very memory-efficient
- This could be improved by re-using a “scratchpad” image, instead of creating and discarding a new tinted image every time



NSColor

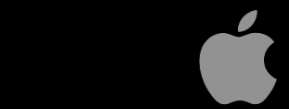
- Represents not only colors, but “meta” colors, “catalog” colors, patterns, . . .
- To paint with an NSColor, send it a **-set** message
- To create an NSColor, use methods such as:

+colorWithPatternImage:, +whiteColor,
+colorWithCalibratedRed: green: blue: alpha:,
+controlHighlightColor

NSImage Pitfalls

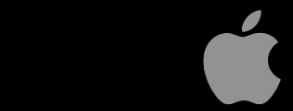
- Check the size!
  - The pixel dimensions are not the same thing as the image size. Use the **-size** and **-setSize:** methods
- Many apps get this wrong, so be alert
- Drawing scaled images

-composite... and **-dissolve...** methods behave differently than **-drawInRect...** methods



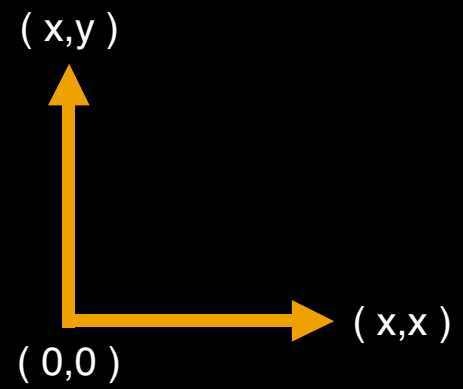
Upcoming Features of NSImage

- Progressive display
  - The delegate will be notified when the size is known, when chunks of pixels arrive, and when the entire image is finished loading
- Support for multi-frame images (animated GIFs)
  - The frames become one long bitmap



The Quartz Coordinate System

- We stole it from René Descartes: the origin is in the lower-left corner



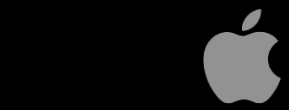
- Coordinates are floating-point values
- Whole-number coordinates land between screen pixels

The Quartz Coordinate System (Cont.)

- If it is inconvenient, change it!
- An **NSView**'s coordinate system can be rotated, scaled, and translated with methods like

  -rotateByAngle:, -scaleUnitSquareToSize:,

  -translateOriginToPoint:, and **-setBoundsRotation:**
- The **NSAffineTransform** class describes any combination of scaling, translation, and rotation



The Frame Rectangle

- The rectangle describing a view's location and size in its *superview's coordinate* space
  - Get it with the **-frame** method
  - Change it with **-setFrame** and **-setFrameRotation:**

The Bounds Rectangle

- Describes the size of the view in its own coordinate space
- Defines the location of the origin
- Change it with **-setBoundsSize:, -setBounds:, -setBoundsRotation:, -setBoundsOrigin: -translateOriginToPoint:, -rotateByAngle: -scaleUnitSquareToSize:,**

NSAffineTransform

- An object that describes operations on a coordinate space
- It has many of the same methods as NSView for changing a coordinate space: **-rotateByDegrees:, -translateXBy:yBy:, -scaleBy:**
- Transform operations can be combined using **-appendTransform:** and **-prependTransform:**

NSAffineTransform (Cont.)

- I have set up my NSAffineTransform, how do I use it?
- Operations on objects and data: **-transformPoint: -transformSize:, -transformBezierPath:**
- Apply it to the current drawing context:

  -set, -concat

Demo

Transformed View

What I Did Wrong in That Demo

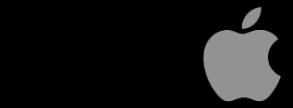
- Drawing too much
  - I could have only drawn the parts that changed, using **-setNeedsDisplayInRect:**

Demo

Transformed Image

What Did I Do Wrong This Time?

- Nothing! It was perfect



NSBezierPath

- This is how you should do most vector-based drawing
- Encapsulates the Quartz 2D drawing model
  - Path construction methods
    -moveTo:, -lineTo:, -curveTo:
    -appendBezierPathWith...
  - Path attributes
    -lineJoin, -lineWidth, -miterLimit,- flatness
  - Drawing the Path
    -fill, -stroke

NSBezierPath (Cont.)

- Convenience methods:

  +strokeLineFromPoint:toPoint:

  +fillRect:, +bezierPathWithOvalInRect:

- Add your own:

  If there is a shape you use a lot, add a method to NSBezierPath to draw it

Demo

StringArt View

What I Did Wrong This Time . . .

- Inefficient drawing!
- Clearing/redrawing the entire bounds
  - I should just clear the rectangle passed to me in **-drawRect:**
  - I did not re-use the bezierpath
- Made lame excuses because it is just a demo

NSBezierPath Considerations

- The rasterizer charges by the intersection
- Take care with pixel placement when stroking a path
- Check whether you are drawing for the screen or the printer

The Graphics State

- Looking for **gsave** and **grestore**?
- You just found them!
- The NSGraphicsState class keeps a stack of graphics states, just like Postscript
- NSGraphicsState methods: **+ saveGraphicsState, + restoreGraphicsState, - setShouldAntiAlias:** **+ currentContextDrawingToScreen,** etc.

Kit Functions and Macros

- Rectangle drawing functions
  - **NSRectFill(), NSRectFillList(), NSRectFillListWithColors(), NSRectFillUsingOperation(), NSRectFillListWithColorsUsingOperation()**
- Geometry Conveniences
  - **NSDivideRect(), NSContainsRect(), NSEqualRects(), NSMakeRect(), NSMouseInRect(), NSHeight(), NSWidth(), NSMinX, NSMaxY()**, and so on . . .
- If you think it should be there, look for it; It probably exists

Drawing Strings

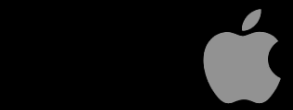
- The AppKit adds drawing methods to NSString:
  -drawAtPoint:withAttributes:,
  -drawInRect:withAttributes:,
  -sizeWithAttributes:
- . . .And to NSAttributedString:
  -drawAtPoint:, **-drawInRect:**, **-size**

Drawing With NSCell

- Cells are objects that draw themselves in Views
- Use a Cell when the overhead of an NSView is not necessary
- The AppKit includes NSCell classes for drawing images and text

Drawing With NSCell (Cont.)

- Use a Cell for drawing lines or blocks of editable text
- Cells can be re-used, as in NSBrowser and NSTableView
- NSControl subclasses typically have a cell do all their drawing



Drawing With NSCell (Cont.)

- A cell is often a good starting point for your custom drawing
- Cells do their drawing in **-drawInRect:inView:**
- In a cell subclass, you can use the inherited drawing behavior

Demo

Cells and Views

Conclusion

- We saw how to draw using the classes in the Cocoa framework
- And how to do some sophisticated things with not much code at all

Cocoa: *Simple things simple,*
Complex things possible

Cocoa Documentation

- Object-Oriented Programming and the Objective-C Language
- Programming Topics
    - Application Architecture
    - Memory Management
    - Foundation Framework
    - Multithreading
    - Loading Resources
    - Notifications

and many more!

Documentation > Cocoa
developer.apple.com/techpubs/macosx/Cocoa/CocoaTopics.html

For More Information

- O'Reilly "Learning Cocoa" and "Building Cocoa Applications: A Step-by-Step Guide"
- Cocoa Developer Documentation

  http://developer.apple.com/techpubs/macosx/Cocoa/CocoaTopics.html
- Apple Customer Training

  http://train.apple.com/

Roadmap

300 Introduction to Cocoa: What is Cocoa?	Room A1 **Mon., 5:00pm**
301 Cocoa: What's New?	Civic **Tues., 9:00am**
303 Cocoa Scripting: Scripting overview and recent changes	Room A2 **Thurs., 10:30am**
304 Cocoa Controls and Cocoa Accessibility Overview of controls; new Accessibility APIs	Room A2 **Thurs., 5:00pm**

Roadmap

305 Cocoa Drawing: Drawing using Cocoa APIs	Hall 2 **Fri., 10:30am**
306 Cocoa Text: In-depth overview of the text system	Room J **Fri., 2:00pm**
FF016 Cocoa: Comments and suggestions for Cocoa	Room A1 **Fri., 5:00pm**

Who to Contact

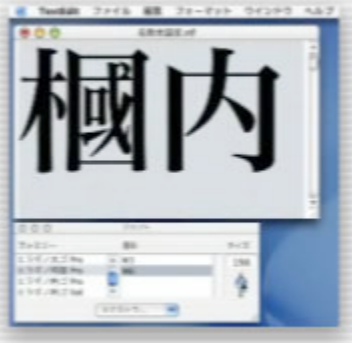
Heather Hickman
Cocoa Technology Manager
Apple Worldwide Developer Relations
hhickman@@apple.com

Cocoa Feedback
cocoa-feedback@group.apple.com

Cocoa Development Mailing List
Subscribe at
www.lists.apple.com/mailman/listinfo/cocoa-dev

http://developer.apple.com/wwdc2002/urls.html

Q&A



Cocoa Frameworks Teams

http://developer.apple.com/wwdc2002/urls.html


WWDC2002


WWDC2002


WWDC2002